EXAMEN DE FIN D'ÉTUDES SECONDAIRES GÉNÉRALES

Sessions 2023 – QUESTIONNAIRE ÉCRIT

Date :	08.06.23	*Durée :*	14:15 - 17:15	*Numéro candidat :*	
Discipline :	Informatique	*Section(s) :*	GIG		

Dans votre répertoire de travail (à définir par chaque lycée), vous trouverez un dossier nommé **EXAMEN_GIG**. Renommez ce dossier en remplaçant le nom par votre numéro de candidat (exemple de notation : **LXY_GIG2_07**). Tous vos fichiers devront être sauvegardés à l'intérieur de ce dossier, qui sera appelé **votre dossier** dans la suite !

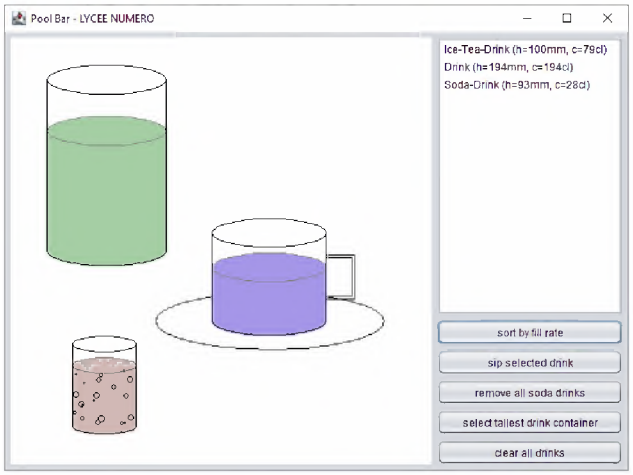
Vous trouvez une version exécutable du programme – ***Demo.jar*** – dans votre dossier.
Avant de commencer, il est recommandé de lancer et d'essayer ce programme.

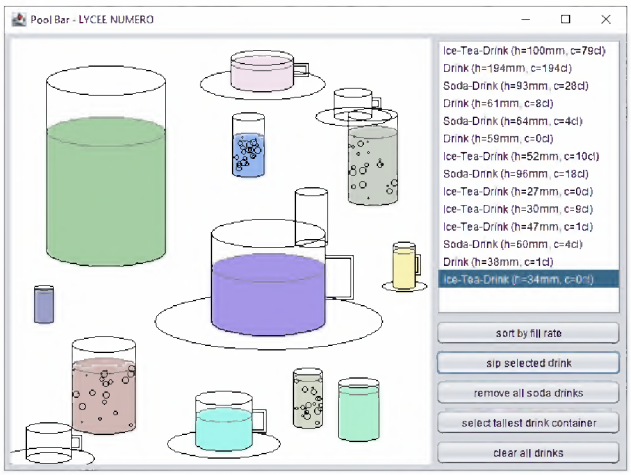
En cas de non-respect de la nomenclature, des conventions du cours, des indentations, des indications de l'UML, etc., jusqu'à 3 points pourront être retranchés de la note finale.

Détente auprès du « PoolBar » (2 + 13 + 4 + 4 + 18 + 3 + 16 = 60 p.)

Développez le projet **PoolBar** qui gère des boissons rafraîchissantes et qui est basé sur les classes **Drink**, **Soda**, **IceTea**, et **Bar**.

Les copies d'écran ci-dessous en montrent deux exemples d'exécution.

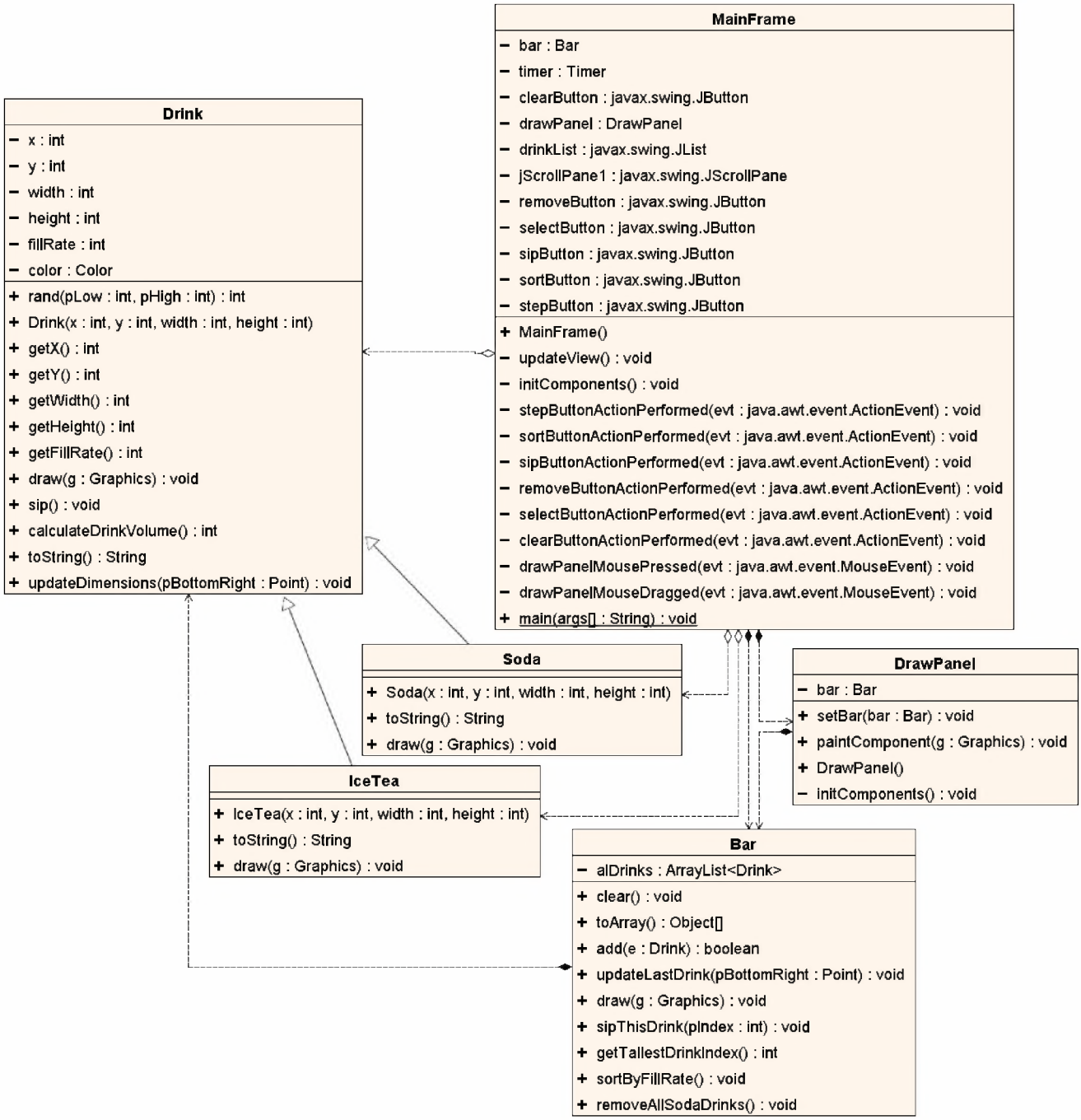




<u>Préliminaire</u> :

- dans le présent projet, ne créez pas d'autres attributs que ceux indiqués dans le questionnaire et illustrés dans le diagramme UML ;
- chaque classe fille doit, pour autant que possible, se servir des méthodes de sa classe mère ;
- ajoutez, tout en haut de chaque classe, votre numéro de candidat en commentaire.

Diagramme UML :



Génération automatique de code (2 p.)

Partie 1 : la classe Drink (1 + 1 + 5 + 1 + 2 + 1 + 2 = 13 p.)

Implémentez la classe **Drink** – avec ses attributs et accesseurs requis – en vous basant sur le diagramme UML donné (voir page 2) et les indications supplémentaires ci-dessous.

La classe **Drink** décrit une boisson, au récipient cylindrique et à base circulaire et dispose des attributs suivants :

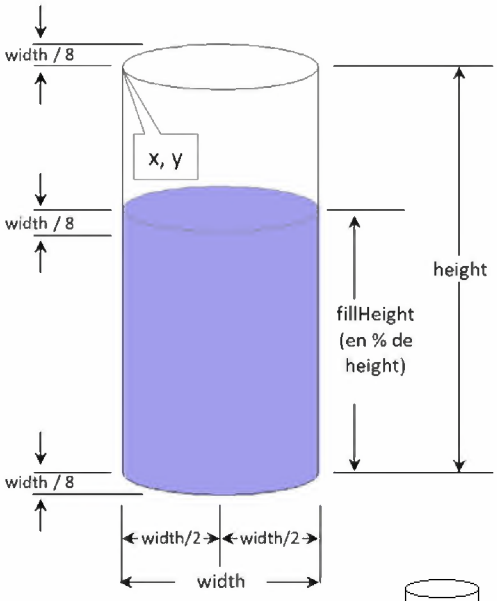
x, y :	les coordonnées du coin supérieur gauche du récipient,
width, height :	les dimensions du récipient (en pixels),
fillRate :	le taux de remplissage (en pourcent),
color :	la couleur du contenu.

La classe **Drink** dispose des méthodes suivantes :

a) **rand** retourne un nombre aléatoire entier ϵ [pLow, pHigh].
Utilisez toujours cette méthode pour calculer un nombre aléatoire ; (1 p.)

b) le **constructeur** initialise
- les attributs à la valeur des paramètres respectifs,
- le taux de remplissage à une valeur aléatoire ϵ [60, 90],
- la couleur à une nuance aléatoire, dont chacune des composantes rouge, verte et bleue est contenue dans l'intervalle [150, 250] ; (1 p.)

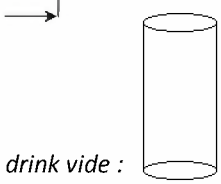
c) **draw** dessine la boisson sur le canevas avec la couleur indiquée et comme illustré sur le dessin ci-contre. Si le verre est vide, alors le fond du récipient est sans couleur et donc transparent.
Le cas échéant, le bord supérieur du contenu est mis en évidence par une ellipse grise ; (5 p.)



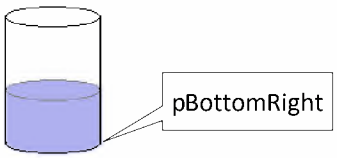
d) **sip** fait boire une gorgée et fait donc réduire le taux de remplissage d'une valeur aléatoire ϵ [5, 15], sans jamais devenir négatif ; (1 p.)

e) **calculateDrinkVolume** admet que la taille d'un pixel équivaut à 1 mm et retourne ainsi le volume du contenu, exprimé en centilitres [cl] et arrondi vers le bas ; (2 p.)

f) **toString** retourne une chaîne indiquant la hauteur (en admettant toujours l'équivalence 1px ≈ 1mm) du récipient et le volume du contenu dans le format :
`Drink (h=xmm, c=ycl)`
p.ex. : `Drink (h=159mm, c=78cl)` ; (1 p.)


drink vide :

g) **updateDimensions** recalcule les dimensions (largeur et hauteur) du verre en fonction du point **pBottomRight** fourni par paramètre, sans que les nouvelles dimensions du récipient ne puissent descendre en-dessous de 20px. (2 p.)



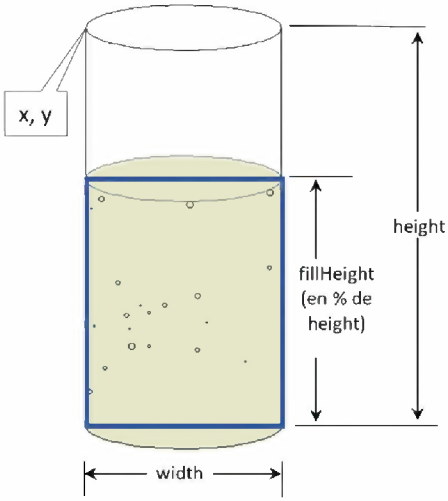
Partie 2 : la classe Soda (1 + 3 = 4 p.)

Implémentez la classe **Soda**, dérivée de la classe **Drink**, en vous basant sur le diagramme UML donné (voir page 2) et les indications supplémentaires ci-dessous.
La classe **Soda** décrit une boisson au gaz carbonique (bulles) et dispose des méthodes suivantes :

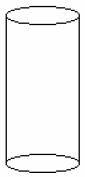
a) **toString** retourne une chaîne similaire à celle de sa classe mère selon le format :
`Soda-Drink (h=xmm, c=ycl)`
p.ex. : `Soda-Drink (h=159mm, c=78cl)` ; (1 p.)

b) **draw** dessine la boisson sur le canevas avec sa couleur.

Au dessin de la classe mère sont ajoutées 20 bulles noires, creuses, de diamètre aléatoire ϵ [2, 7] qui ne doivent pas sortir (même pas partiellement) du rectangle bleu (voir dessin ci-contre). Ce rectangle bleu n'est pas à dessiner. (3 p.)



soda vide :



Partie 3 : la classe IceTea (1 + 3 = 4 p.)

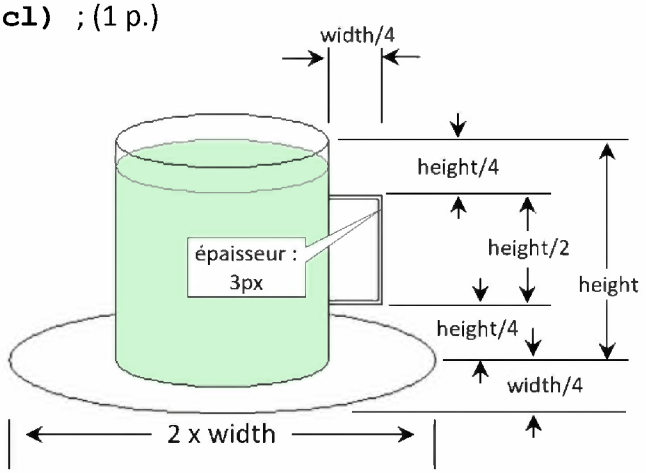
Implémentez la classe **IceTea**, dérivée de la classe **Drink,** en vous basant sur le diagramme UML donné (voir page 2) et les indications supplémentaires ci-dessous.
La classe **IceTea** décrit un thé froid et dispose des méthodes suivantes :

a) **toString** retourne une chaîne similaire à celle de sa classe mère selon le format :
`Ice-Tea-Drink (h=xmm, c=ycl)`
p.ex. : `Ice-Tea-Drink (h=89mm, c=64cl)` ; (1 p.)

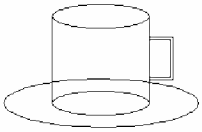
b) **draw** dessine la boisson sur le canevas avec sa couleur.

Au dessin de la classe mère sont ajoutées une soucoupe en bas et, sur le côté droit, une anse d'une épaisseur de 3px et d'une hauteur totale de **height/2**.

Veillez à dessiner les différents éléments dans le bon ordre. (3 p.)



ice tea vide :



Partie 4 : la classe Bar (1 + 1 + 1 + 5 + 6 + 4 = 18 p.)

Implémentez la classe **Bar** – avec son attribut et ses méthodes requis – en vous basant sur le diagramme UML donné et les indications supplémentaires ci-dessous.

La classe **Bar** gère une liste de boissons à l'aide du seul attribut **alDrinks**, dispose des méthodes standard **add**, **clear** et **toArray** et des méthodes suivantes :

a) **updateLastDrink** met à jour les dimensions de la dernière boisson de la liste ; (1 p.)

b) **draw** dessine toutes les boissons de la liste sur le canevas ; (1 p.)

c) **sipThisDrink** – après vérification de l'indice fourni - fait boire une gorgée de la boisson d'indice donné ; (1 p.)

d) **getTallestDrinkIndex** retourne l'indice du récipient le plus haut de la liste, **-1** si la liste est vide ; (5 p.)

e) **sortByFillRate** trie avec l'algorithme de tri par sélection directe (par ordre croissant) les boissons de la liste selon leur taux de remplissage ; (6 p.)

f) **removeAllSodaDrinks** supprime toutes les boissons de type **Soda** de la liste. (4 p.)

Partie 5 : la classe DrawPanel (3 p.)

Implémentez la classe **DrawPanel** – avec son attribut et son manipulateur requis – en vous basant sur le diagramme UML donné et l'indication supplémentaire ci-dessous.

La méthode **paintComponent** dessine, si possible, toutes les boissons du bar sur un fond blanc.

Partie 6 : la classe MainFrame (4 + 1 + 1 + 1 + 1 + 2 + 1 + 1 + 1 + 2 + 1 = 16 p.)

Implémentez la classe **MainFrame** – avec ses attributs requis – en vous basant sur le diagramme UML donné et les indications supplémentaires ci-dessous.

La classe **MainFrame** dispose des attributs suivants :

bar le bar, instancié dès le départ ;

timer le chronomètre.

Reproduisez fidèlement l'interface de la classe **MainFrame**, illustrée ci-dessous et respectez la nomenclature indiquée. À titre d'indication, la largeur du panneau de dessin est de 475 px et celle de la liste de 210 px.

Inscrivez « **Pool Bar** » suivi de votre **numéro de candidat** dans l'entête de la fiche. (4 p.)

La classe **MainFrame** dispose des méthodes suivantes :

a) le **constructeur** effectue les initialisations nécessaires et crée, puis lance une instance de chronomètre, associé au bouton caché **stepButton** et à raison de 10 appels par seconde ; (1 p.)

b) **updateView** met à jour toute l'interface. Utilisez cette méthode pour garder à jour votre interface ; (1 p.)

c) **stepButtonActionPerformed** fait redessiner le panneau de dessin ; (1 p.)

d) **sortButtonActionPerformed** fait trier les boissons du bar ; (1 p.)

e) **sipButtonActionPerformed** fait boire une gorgée de la boisson sélectionnée dans le bar. Après cette action, la même boisson restera toujours sélectionnée ; (2 p.)

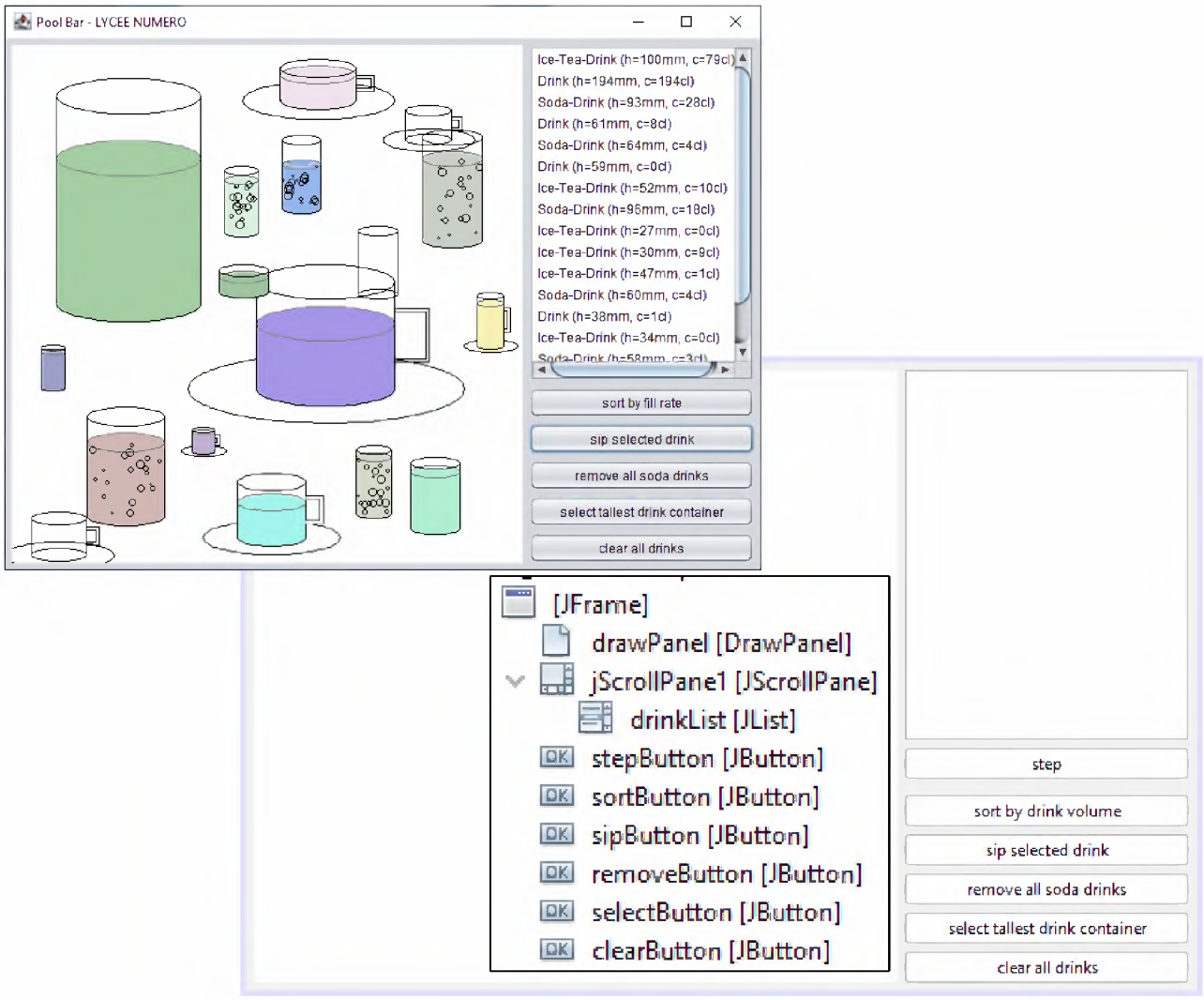
f) **removeButtonActionPerformed** supprime toutes les boissons de type **Soda** du bar ; (1 p.)

g) **selectButtonActionPerformed** sélectionne le récipient le plus haut du bar ; (1 p.)

h) **clearButtonActionPerformed** supprime toutes les boissons du bar ; (1 p.)

i) la **pression d'un bouton de la souris** sur le panneau **drawPanel** crée une nouvelle boisson à cet endroit et l'ajoute au bar. La nouvelle boisson aura une taille de 20px sur 20px par défaut. Selon le bouton de la souris on distingue le type de boisson (gauche = Drink, milieu = IceTea, droit = Soda) ; (2 p.)

j) un **déplacement avec un bouton appuyé** fait redimensionner la dernière boisson ajoutée en fonction des coordonnées de la souris. (1 p.)



Enseignement secondaire général
Division technique générale – Section technique générale
Examen 1GIG

Liste des composants et classes connus

Liste des composants (propriétés, événements et méthodes) et classes à connaître pour l'épreuve en informatique à l'examen de fin d'études secondaires générales - division technique générale.

Package	Classe	Details	Remarques / Constantes
javax.swing	JFrame	Méthodes - setTitle(...) / getTitle() NetBeans Object Inspector Property - title	
	JButton JLabel JTextField	Méthodes - setText(...) / getText() - setVisible(...) - setEnabled(...) Événement - actionPerformed NetBeans Object Inspector Property - icon	- le libellé *JLabel* peut aussi être utilisé pour visualiser des images via la propriété « icon » de l'inspecteur objet de NetBeans (le composant *JTextField* ne possède pas de propriété « icon »).
	JSlider	Méthodes - setMinimum(...) / getMinimum() - setMaximum(...) / getMaximum() - setValue(...) / getValue() Événement - stateChanged	
	JPanel	Méthodes - setVisible(...) - setBackground(...) / getBackground() - getWidth() / getHeight() - paintComponent(Graphics g) - repaint() Événements - MousePressed / MouseReleased - MouseDragged / MouseMoved	- *JPanel* est utilisé pour regrouper d'autres composants visuels et pour réaliser des dessins. - Lors de la réalisation de dessins, la méthode **`public void paintComponent (Graphics g)`** est à surcharger.
javax.swing	JList	Méthodes - setListData(...) - getSelectedIndex() / setSelectedIndex(...) Événement - valueChanged NetBeans Object Inspector - *Properties* - model - selectionMode (SINGLE) NetBeans Object Inspector - *Code* - Type Parameters : - *(vide)*	- *JList* est utilisé surtout pour afficher le contenu d'une liste *ArrayList*.
java.awt.event	ActionEvent	- Ce type d'objet est uniquement utilisé dans les méthodes de réaction ajoutées de manière automatique à l'aide de NetBeans.	
	MouseEvent	Méthodes - getX() / getY() - getPoint() - getButton()	Constantes - BUTTON1 - BUTTON2 - BUTTON3

Package	Classe	Details	Remarques
javax.swing	Timer	Constructeur - Timer(int,ActionListener) Méthodes - start() - stop() - setDelay(...) - isRunning()	
		- Comme *ActionListener* on utilisera de préférence celui d'un bouton. Exemple : `timer = new Timer(1000,stepButton.getActionListeners()[0]);`	
java.awt	Graphics	Méthodes - drawLine(...) - drawOval(...) / fillOval(...) - drawRect(...) / fillRect(...) - drawString(...) - setColor(...) / getColor()	
	Color	Constructeurs - Color(...)	
	Point	Constructeurs - Point(...) Attributs (publics) - x et y Méthodes - getLocation() / setLocation(...)	
java.util	ArrayList	Méthodes - add(...) - clear() - contains(...) - get(...) - indexOf(...) - remove(...) - set(...) - size() - isEmpty() - toArray()	- `Object[] toArray()` est employé uniquement pour passer les contenus d'une liste à la méthode *setListData(...)* d'une *JList*.
java.lang	String	Méthodes - equals(...) / compareTo(...) - contains(...) - valueOf(...)	
	Integer Double	Méthodes - equals(...) / compareTo(...) - valueOf(...)	
	Math	Méthodes - abs(...) - round(...) - random() - sqrt(...) - pow(...) - sin(...), cos(...), tan(...)	Constante: - PI
	System	Méthode - out.print() - out.println()	